Physically Based Rendering (4th Ed): Chapter 6 Summary

Shapes: Intersection and Differential Geometry

Gemini (For a Developer Audience)

November 29, 2025

Introduction: Hitting Things

Chapter 6 is the "Collision Detection" chapter. We have a Ray: $R(t) = O + t\mathbf{d}$. We have a Shape (Sphere, Triangle, Cylinder). We need to find t such that the ray touches the shape.

But PBRT goes deeper. It's not enough to just hit the shape. We need to know *everything* about the surface at that hit point: the normal, the UV coordinates, the tangent vectors, etc. This is called **Differential Geometry**.

1 The Basic Intersection Equation

1.1 The Sphere

A sphere at the origin with radius r is defined by:

$$x^2 + y^2 + z^2 - r^2 = 0$$

Substitute the Ray equation ($x = O_x + td_x$, etc.) into the Sphere equation:

$$(O_x + td_x)^2 + (O_y + td_y)^2 + (O_z + td_z)^2 - r^2 = 0$$

This expands into a standard Quadratic Equation:

$$At^2 + Bt + C = 0 \tag{1}$$

Where:

- $A = \mathbf{d} \cdot \mathbf{d}$ (Length of direction vector squared, usually 1).
- $B = 2(\mathbf{O} \cdot \mathbf{d})$.
- $C = \mathbf{O} \cdot \mathbf{O} - r^2$.

The Solution: Use the Quadratic Formula: $t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$.

- If $B^2 - 4AC < 0$: No intersection (Ray misses).
- If $B^2 - 4AC > 0$: Two intersections (Entry and Exit). Take the smallest positive t.

1.2 The Triangle

Triangles are the most common shape. PBRT uses the **Möller-Trumbore algorithm**. It uses Barycentric Coordinates. Any point P inside a triangle (A, B, C) can be written as:

$$P = (1 - u - v)A + uB + vC$$

We equate the Ray to the Triangle:

$$O + t\mathbf{d} = (1 - u - v)A + uB + vC$$

This gives us a system of linear equations to solve for t, u, and v. If $u \geq 0$, $v \geq 0$, and $u + v \leq 1$, the hit is valid.

2 Differential Geometry: Knowing the Surface

Once we hit a surface, we populate a ‘SurfaceInteraction‘ object. This is crucial for shading.

2.1 1. The Normal (n)

The vector perpendicular to the surface.

- **Sphere**: The normal at point P is simply Normalize$(P - Center)$.
- **Triangle**: The cross product of two edges. $(B - A) \times (C - A)$.

The normal tells us which way the surface is facing, which determines how light reflects.

2.2 2. Parametric Coordinates (u, v)

Every surface in PBRT is parameterized by (u, v) coordinates.

- Used for **Texture Mapping**.
- On a sphere, u maps to longitude (ϕ) and v maps to latitude (θ).

2.3 3. Partial Derivatives $(\frac{\partial P}{\partial u}, \frac{\partial P}{\partial v})$

This is the calculus part.

- $\frac{\partial P}{\partial u}$: If I move slightly in the u direction on the texture, which way does the point P move in 3D space?

- $\frac{\partial P}{\partial v}$: Same for v.

These two vectors are **Tangent** to the surface. The Cross Product of these tangents gives us the Normal:

$$\mathbf{n} = \text{Normalize}\left(\frac{\partial P}{\partial u} \times \frac{\partial P}{\partial v}\right)$$

Why do we need Derivatives?

Why not just store the Normal? Because of **Bump Mapping** and **Anisotropy**.
If you want to simulate brushed metal (Anisotropy), the reflection depends on the direction of the scratches. The derivatives $\frac{\partial P}{\partial u}$ tell us exactly which way the "scratches" (texture coordinates) flow across the 3D surface.

3 Managing Rounding Error

Computers use floating-point numbers (floats). Floats are not real numbers; they have gaps. When a ray hits a surface at t, the calculated point $P = O + t\mathbf{d}$ might be slightly *below* the surface due to rounding error.

Self-Intersection (Shadow Acne): If you spawn a new shadow ray from P, it might immediately hit the surface itself because P is technically "inside" the object. The surface shadows itself, creating ugly black speckles.

The Solution: Robust Spawn PBRT uses a robust method to spawn rays. Instead of starting exactly at P, it offsets the origin slightly along the Normal $\mathbf{n}$.

$$O_{new} = P + \epsilon\mathbf{n}$$

PBRT calculates this ϵ (epsilon) dynamically based on the magnitude of the coordinates, ensuring the ray definitely clears the surface boundary.

Summary for the Developer

1. **Ray-Shape Intersection**: It's just algebra. Sphere = Quadratic Formula. Triangle = Linear System.
2. **SurfaceInteraction**: The result of an intersection isn't just a point. It's a data structure containing Position, Normal, UVs, and Tangents (Derivatives).
3. **Tangents matter**: You need partial derivatives ($\partial P/\partial u$) to do advanced shading like bump mapping or brushed metal.
4. **Float Precision**: Never trust 'float'. Always offset your rays slightly to avoid self-intersection artifacts.